Introduction to SMOK

Due November 5th, 2001

Purpose:

This assignment is where the pieces of the CPU puzzle come together for you: instructions, data path, control. This is the great leap of this course, and once past it we go on to ever more exciting things ("to infinity and beyond!"). It is also an introduction to the SMOK design tool. SMOK allows you to design and test architectures at a component abstraction level. Finally this assignment requires you to write a procedure in MIPS assembly [actually hand assemble it to MIPS machine code], so the mechanics of procedure stack frames is also a component of this assignment.

Task:

Task 1: You are to build a single-cycle small-scale MIPS microprocessor. This will be just like a commercial MIPS microprocessor (R2000 vintage) except that you will implement only a hand full of instructions, and you won't have to pipeline the machine (although you may think about how to do that since HW5 is coming up ;-), everything will happen in one huge (long) clock cycle.

Task 2: Write a recursive function that calculates the n'th Fibonacci number. This function Fibonacci(i) will return an integer. You are to write this function first in MIPS assembly language, and then you are to hand assemble your code into binary form. You will then execute this code on the machine you build in Task 1. Because of this you will have to write this function using only the set of instructions you actually implement in Task 1, but these should be enough.

Details:

a) You may (are encouraged!) to work in pairs.

b) Use the "turnin" command to hand in this assignment. To do this, send all of your files to one of the undergraduate instructional machines and type "turnin -ccse378 -phw4 file1 file2..."

c) We are going to use the latest version of SMOK, which executes on a PC. This version can be downloaded from "http://www.cs.washington.edu/homes/zahorjan/homepage/Tools/SMOK/". Please use the latest version since it has been updated specifically for this class.

d) The format of the instructions for your machine is the real MIPS encoding format (so R-type, I-type and J-type). In addition, just like the Real McCoy, your machine will have 32 integer registers, with register zero hardwired to zero.

e) The absolute reference for any details with this assignment is the real MIPS documentation, and any other documentation on the real MIPS microprocessor [unless otherwise explicitly modified in this handout, or from subsequent announcements]. Having said this, we are going to make a modification to the memory layout of the process the MIPS processor you build executes. This is only a small hack to make things easier for the SMOK tool and yourselves, but it should be easy to see how to modify things to work like the real microprocessor. The hack is as follows:

> The real MIPS processor has a protected and privileged operating mode. This is to support process level protection mechanisms that are conventional on modern operating systems. For this assignment there is no operating system, and no protection modes.
>
> While your microprocessor operates with 32 bit addresses, in order to not make SMOK explode you are going to work with only a 256 word (1024 byte) memory.
>
> Conventionally a process starts at 0x04000000. However, your initial PC should be set to 0x00000000. Your code should start at address 0.
>
> Conventionally a process stack pointer is set to 0x7ffxxxxx (near the 2 gigabyte boundary). You should set your stack pointer at 1024 bytes (256 words) (i.e. just at the end of the memory in the machine)
>
> Conventionally a process stores static and dynamic data starting at 0x10000000. You should start storing static and dynamic data at the 512 byte (128 word) boundary.
>
> The real MIPS processor contains delay slots behind branches and delays on load instructions (to account for the memory access). Your processor will operate in a single cycle, and so there will be no delay slots.

f) You should hand assemble the following code into binary form and place it at address 0:

```
0x00000000:  ori     $sp, $0, 1024   # effectively a li $sp, 1024
0x00000004:  addiu   $sp, $sp, -4    # effectively a sub $sp, $sp, 4
0x00000008:  ori     $a0, $0, 10     # effectively a li $a0, 10
0x0000000C:  jal     fibonacci
0x00000010:  addiu   $sp, $sp, 4
0x00000014:  beq     $0, $0, -4      # effectively a halt
0x00000018:  ---- place your fibonacci program here
```

This code sets up the stack pointer, and then makes a procedure call into the fibonacci procedure that you are to write. After returning from fibonacci it removes the stack frame and halts the machine by having a branch instruction jump to itself.

g) You must support the following instructions:

Arithmetic:

- ADDU, ADDIU
- SUBU

SLTIU, SLTU
ORI, OR

Load/Store:
LW, SW

Branch:
JAL, JR
BEQ, BNE

h) Fibonacci numbers are a rather famous sequence of numbers that go something like this:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

The basic pattern is F(n) = F(n-1) + F(n-2) [with F(0)=0 and F(1)=1]. So your function should be like this:

```
Fibonacci(int   n)
        {
        if ( n == 0)      return(0)
        if ( n <=2 )      return ( 1 );
        return ( Fibonacci(n-1) + Fibonacci(n-2) );
        }
```

Note that the function is recursive so you will need to be careful about how you handle registers. Follow MIPS conventions.

Remember to use only the instructions your machine supports. It should be more than enough.

Do not worry about improper inputs.

Please do not be too fancy in how you implement this function. We have to grade it!

i) It will behoove both of you (if your working in pairs) to work on all parts of this assignment. I.e., you can expect me to ask questions on the midterm/final about all aspects of all assignments.

Notes:

- To know SMOK is to love SMOK. But, just like Al Capone when he said the secret to winning elections is to "vote early and often", the secret to loving SMOK is to "save early and often".

- This site is your friend:
www.cs.washington.edu/homes/zahorjan/homepage/Tools/SMOK/ComponentRef/index.htm

- You will want to understand what a container is and how to use one.